Lenguajes de Programación

Práctica 8 - Macros y paradigma orientado a objetos

Semestre 2017-1

Fecha de inicio: 18 de noviembre de 2016

Fecha de término: 9 de diciembre de 2016

Instrucciones

- Resolver los siguientes ejercicios de manera clara y ordenada.
- Se deben modificar y/o enviar los archivos `macros.rkt` más los archivos necesarios de la parte II.
- La entrega es en equipos de máximo 3 integrantes. Seguir los lineamientos especificados en: `http://sites.ciencias.unam.mx/ldp171/formato`.
- Los puntos extra son individuales y sólo se puede escoger uno. El punto extra teórico debe de realizarse **a computadora y entregarse impreso en hojas recicladas a más tardar el lunes 5 de diciembre de 11 a 13 en el salón O127**. El punto extra de programación se envía por correo en un archivo `<no.cuenta>_P08.rkt` enviar con el asunto `[LDP-EXP8]`. Incluir el nombre completo del autor en el cuerpo del correo.

Parte I: Macros

1. **Estructura de repetición `for`**

   Definir un macro que genere la sintaxis necesaria para dar el comportamiento a una estructura de repetición `for`. El macro `for` debe tener la siguiente forma que representa las partes de toda estructura de control:

   ```
   (for i c a do body)
   ```

   - El parámetro `i` representa la inicialización de la variable de control. Para facilitar la implementación se supondrá que la inicialización se hace fuera del `for`.
   - El parámetro `c` representa la condición.
   - El parámetro `a` representa la actualización a la variable de control haciendo uso de la primitiva `set!`.
   - El parámetro `body` representa las instrucciones que se deben ejecutar en cada instrucción. El número de instrucciones puede ser uno o más.

   Nota: `do` es una palabra reservada de la estructura `for`.

   Ejemplo:

```
> (define i 0)
> (for i (< i 3) (set! i (+ i 1)) do
   (displayln ‘‘Hola’’)
   (displayln i))
Hola
Hola
Hola
```

2. **Autómata**

   En clase se definió un macro para representar un autómata finito determinista (capítulo 37 de "*Programing Languages: Aplications and Interpretation*") y se usó para determinar si una expresión pertenecía a una gramática. Usando el macro definido en clase, construir un autómata que acepte expresiones de la siguiente gramática:

```
<expr> ::= <id>
         | <num>
         | <bool>
         | <list>
         | {with {{<id> <expr>}+} <expr>}
         | {rec {{<id> <expr>}+} <expr>}
         | {fun {<id>*} <expr>}
         | {if <expr> <expr> <expr>}
         | {<op> <expr>+}
         | {<expr> <expr>*}

<id> := a | .. | z | A | ... | Z | aa | ab | ... | aaa | ...
        (Cualquier combinación de caracteres alfanuméricos
              con al menos uno alfabético)

<num> ::= ... | -2 | - 1 | 0 | 1 | 2 | ...

<bool> ::= true | false

<list> ::= empty
         | {cons <expr> <expr>}

<op> ::= + | - | * | / | % | min | max | pow | number?
       | and | or | not | < | > | <= | >= | = | != | zero?
       | head | tail | empty? | list?
```

Parte II: Paradigma orientado a objetos

Al inicio del curso vimos que un analizador léxico es un programa que transforma una cadena de entrada en una lista de lexemas que son elementos finales de una gramática dada. Hay varias maneras de implementar un analizador léxico. En esta práctica se implementará por medio de un autómata finito determinista.

Para esta implementación se construye un autómata finito que reconozca todas las posibles cadenas que forman los lexemas. Después se utiliza la noción de aceptación de este autómata para determinar si una cadena dada es un lexema o no.

1. **Analizador léxico**

   Usando un lenguaje orientado a objetos, implementar un analizador léxico que tome una cadena y devuelva una lista de lexemas basada en el autómata del ejercicio I.2, para esto se necesita implementar el autómata en código, en particular implementar la función de transición y ejecutar el autómata por cada palabra de la cadena original.

   El analizador léxico debe poder recibir cadenas desde la terminal (interactuando con el usuario) o a partir de un archivo de texto.

Restricciones generales

- Para implementar el analizador léxico pueden usar el lenguaje orientado a objetos de su preferencia. Se sugieren: Java, Python o Ruby. **Si es un lenguaje nuevo o multiparadigma, preguntar primero al ayudante de laboratorio**.

  Si usan Java, es preferible que compilen y ejecuten el programa con Apache Ant.

- Se deben usar las principales características del paradigma orientado a objetos: Definición de clases y objetos, Herencia, Polimorfismo, etc.

- El formato de salida del programa queda a decisión del equipo, pero siempre deben devolver una lista de lexemas. Se tomará en cuenta la creatividad.

- El achivo readme debe realizarse **en formato PDF** y deben incluir:

  * Una imagen del autómata que acepta las cadenas.
  * Una descripción de cómo se implementó el analizador léxico.
  * Describir las complicaciones y beneficios que trajo el implementar el analizador léxico usando un lenguaje orientado a objetos.
  * Describir con lujo de detalle cómo debe ejecutarse su programa, qué lenguaje de programación se usó y cómo se compila. **Se restarán puntos a la calificación si esta parte no es clara**[1].

Puntos extra

- (1.5 pts.) Leer la entrada "*Goodbye, Object Oriented Programming*" del blog de Charles Scalfani disponible en `https://goo.gl/yZO9os` y escribir una crítica de al menos dos cuartillas indicando aquellas secciones que consideren importantes de resaltar, aquellas con las que no están de acuerdo, las ideas con las que sí y por qué. Deben justificar cada idea usando todo lo que se ha visto durante el semestre en el curso de Lenguajes de Programación.

- (1.5 pts.) Con base en el macro para generar autómatas finitos deterministas definido en el capítulo 37 de "*Programing Languages: Aplications and Interpretation*" modificarlo para que genere ahora autómatas finitos no deterministas y justificar por qué lo son.

[1] El ayudante de laboratorio decidirá arbitrariamente qué es "clara".